Université de Nice Sophia Antipolis

Institut National de Recherche en Informatique et Automatique

Rapport de stage

Études et développement de diagrammes de décisions linéaires

Stagiaire

Havyarimana Dorine

Filiaire

Master II Électronique, Systèmes et Télécommunications

Sous la direction de :

Annie Ressouche

Daniel Gaffé

10 Avril 2017 — 29 Septembre 2017

Pour Carmen et Paulin...

"Empower yourself with a good education, then get out there and use that education to build a country worthy of your boundless promise." – Michelle Obama

Résumé

La vérification de modèle, plus communément appelé Model Checking, est un concept basé sur une approche automatique de vérification formelles des propriétés temporelles sur des systèmes réactifs. INRIA en collaboration avec le LEAT ont développé CLEM, un outil de modélisation et de vérification de propriétés, s'appuyant sur une représentation d'état en automates finis générés automatiquement et représentés par des Diagrammes de Décisions Binaires. Dans une optique d'évolution, le travail effectué durant ce stage a été de développer la bibliothèque de diagramme de décision linéaire, nous nous sommes concentrés sur l'inclusion de nouvelles méthodes de réduction dans les cas d'implication forte et faible. L'objectif de ce travail est de développer la partie vérification de CLEM en remplaçant la représentation actuelle des valeurs fondamentales qui utilisent des diagrammes de décisions binaires(BDDs) par les diagrammes de décisions linéaires(LDDs) ce qui nous permettrait de représenter les états par des valeurs entières et non par des signaux non comparables entre eux. Cette nouvelle bibliothèque de LDDs, une fois implémentée sur CLEM, permettra de faire des vérifications de modèles plus fines et, potentiellement, le rendra plus performant.

Abstract

Model verification, more commonly known as Model Checking, is a concept based on an automatic formal verification approach of temporal properties on reactive systems. INRIA in collaboration with LEAT developed CLEM, a modeling and property verification tool, based on a state representation in finite automata generated automatically using binary decisions diagrams. From an evolutionary point of view, the work carried out during this internship was to develop the library of linear decisions diagrams, we focused on the implementation of new reduction methods in cases of "Imply High" and "Imply Low" case. The objective of this work is to develop the verification part of CLEM by replacing the representation of the fundamental values using binary decisions diagrams(BDDs) with linear decisions diagrams(LDDs) which will allow us to represent the states by integer values instead of signals which are not comparable among themselves. This new library, once implemented on CLEM, will make checks of finer models and, we hope, will make it more powerful.

Remerciements

Je voudrais remercier mes encadrants Annie Ressouche et Daniel Gaffé qui m'ont permis de faire ce stage, je n'oublierai jamais ces 6 mois de travail, de surpassement de soi. Vous m'avez offert la connaissance, c'est un trésor que je chérirai et entretiendrai tout au long de ma vie.

A tout les membres de l'équipe STARS, en particulier les stagiaires, nous avons bien travaillé ensemble mais nous avons également profité de moments d'échanges et de complicité autour de dîners et pique-niques magnifiques.

✳ Mon moment préféré : les feux d'artifice à Cannes ! (Joyeux anniversaire Inès !) ✳

❤❤❤

A ma famille de cœur de France, d'Algérie, du Rwanda, du Burundi.

A Séverine et sa maman pour leur soutien moral, une vie ne suffirait pas pour leur rendre la pareille.

A Carmelle, Dorelle, Lolita, Lynda, Marlyne, Véra mes amies de toujours.

A Khélia, qui a toujours été là pour moi durant nos années en Algérie, j'espère qu'on se reverra bientôt et que la médecine te rend heureuse.

✍✍✍

Cette page ne suffirait pas pour exprimer ma gratitude envers cette personne qui a rendu mon monde si loin de tout les êtres qui me sont chères aussi incroyable et peuplé de magnifiques aventures. Merci à toi Lyes Khacef, le Kabyle partisan de la liberté, de l'expression de soi ; bref, de la vérité. Tu accompliras des choses extraordinaires.

❤❤❤

A Mathys et ses parents Claire et Didi, Franco mon cousin pour leur courage qui me rappelle à chaque fois que rien n'est impossible. Merci pour votre soutien.

A mes parents Césarie et Pierre, mon frère Elvis et mes sœurs Carmen et Marlyn, je vous aime.

✍

Table des matières

Table des figures

Liste des tableaux

List of Algorithms

Chapitre 1

Introduction générale

L'informatique des capteurs dite aussi informatique omniprésente trouve son origine dans une génération naissante d'appareils électroniques publics équipés de capteurs qui est, et sera encore plus, en mesure d'observer, analyser et interpréter les informations reçues par ces capteurs.

Les systèmes réactifs, autrement dit les systèmes dont l'état et le comportement change en fonction de l'évolution temporelle des entrées, sont des systèmes qui se trouvent représenter aussi bien en Science de la Vie qu'en Science de l'Ingénieur, de la Matière et même en Sciences Humaines ; d'intenses recherches expérimentales et théoriques ont vu le jour afin de comprendre et prédire leur évolution.

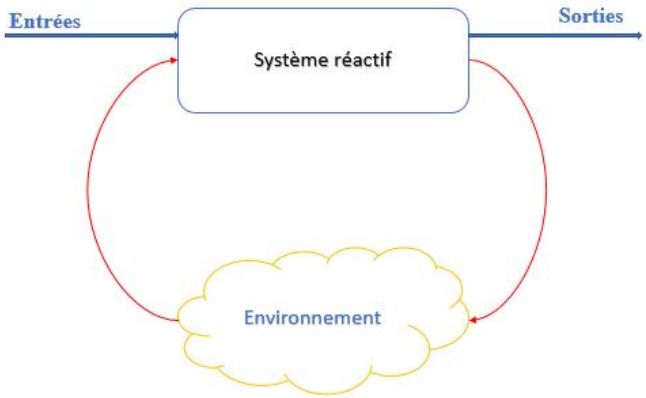

FIGURE 1.1: Système réactif

La caractéristique principale de ces systèmes est d'être capable de définir un nombre de comportements infini grâce aux multiples interactions des éléments qui le composent ; ce qui rend leur conception et mise en œuvre extrêmement difficile. De tels systèmes ont un rôle assez souvent critique (centrales nucléaire, poste de contrôle avionique) qui oblige les concepteurs à passer par une phase coûteuse mais indispensable de test pour éviter d'éventuels dysfonctionnements du système et des exemples sont donnés ici pour témoigner de l'importance de la phase de vérification :

1. **Le Missile de défense Patriot rencontre un problème de logiciel qui conduit à l'échec du système [Arn91]**

   Un calcul inexact du temps en raison d'erreurs arithmétiques informatiques a causé la mort de 28 soldats et blessé 100 autres en Arabie Saoudite.

2. **Destruction du vol 5.01 d'Ariane en 1996 [La96]**

   L'incident a été provoqué par une erreur dans le logiciel de contrôle qui a effectué une conversion de données à partir d'un système flottant de 64 bits en un entier signé de 16 bits, faisant croire à l'ordinateur de bord qu'il fallait inverser la poussée ! Cet incident a coûté 500 Millions de dollars...

3. **Rappel de centaines de milliers de Pentium Intel en 1994**

   Une erreur de conception dans la gestion de la retenue de la division flottante a conduit Intel à inventer le premier processeur qui calcule faux...

Pour des systèmes réactifs dits critiques, l'erreur est interdite (contrôle d'une centrale nucléaire, télé-paiement par carte à puce, etc.). Approcher la réalité des phénomènes temporels de ces systèmes est un vrai challenge scientifique qui, associé au progrès technologiques réalisés au cours de ces dernières décennies ainsi que l'apparition de capteurs dit «intelligents», engendre un besoin de développer des outils de vérification formelles permettant une représentation des systèmes complexes pouvant répondre aux impératifs de sécurité.

Les systèmes critiques qu'ils soient hardware ou software peuvent bénéficier d'une phase de validation sous différentes méthodes :

— La simulation
— Les tests
— La vérification formelle basée sur les méthodes déductives
— La vérification formelle basée sur le Model Checking
— La vérification formelle basée sur l'interprétation abstraite

Les méthodes de tests et de simulation sont les plus classiquement utilisées mais elles présentent un inconvénient majeur, car il est souvent impossible de tester tous les états possibles d'un système : l'ensemble des cas à tester doit être minutieusement choisi afin de couvrir le maximum de scénarios possibles.

La méthode de vérification formelle repose sur une représentation des états possibles du système par le biais d'outils mathématiques. Elle propose une approche rigoureuse de la programmation où l'on a :

— Une représentation mathématique du système.
— Une propriété du système à vérifier.
— Un algorithme de vérification du système.

Tout ceci dans l'objectif de prouver les propriétés fondamentales de :

— **Sûreté** c'est à dire que jamais rien de nuisible au système ou à l'environnement humain n'arrive !
— **Vivacité** c'est à dire que le programme effectue les taches qui lui sont confiées de manière prédictible. En d'autres termes, indépendemment des états précédents du système, si celui-ci arrive dans un état donné et reçoit une entrée donnée, son évolution sera toujours prédictible.

La production de systèmes logiciels fiables constitue l'une des préoccupations majeures des informaticiens qui, pour faire face à la grande évolution technologique de ce siècle, doivent développer des méthodes et outils de vérification de systèmes encore plus performants. La

vérification formelle par Model Checking est l'une des méthodes les plus utilisées car elle permet une vérification simple, efficace, exhaustive mais surtout complètement automatisable. Ce qui la rend avantageuse par rapport aux autres méthodes.

Afin d'être intégré à l'outil de vérification de CLEM, un modèle mathématique des programmes s'appuyant sur une bibliothèque de diagrammes de décisions binaires est en développement sous la forme de diagrammes de décisions linéaires. L'objectif de ce travail est de contribuer à l'amélioration de la bibliothèque de diagrammes de décisions linéaires, qui sera par la suite intégrée dans le compilateur CLEM.

1.1 Vue d'ensemble du sujet de recherche

Ce stage a été effectué au sein de l'INRIA dans le cadre du projet Bio-informatique NeuComp, qui a pour objectif de modéliser et vérifier les propriétés temporelles du modèle *Leaky Integrate and Fire [LIF]*, un modèle de neurone à Spike. Un travail de recherche [DMMG+16] avait été effectué par mes encadrants afin de modéliser et vérifier un réseau de neurones en utilisant le langage synchrone Lustre. Ce travail a permit de représenter des architectures neuronales ainsi que la vérification de quelques propriétés par Model Checking.

CLEM [GR13], un compilateur en langage synchrone Light Esterel[RGR08] développé à l'origine dans l'objectif de synthétiser des automates explicites en une machine implicite de Mealy ou Moore, est utilisé pour interpréter les systèmes réactifs en un modèle bien structuré et facilite l'analyse et le test dans la suite afin de simplifier le travail des Model Checkers qui prouvent des propriétés de logique temporelle sur un modèle du programme représenté sous forme d'équations booléennes et de registres.

Le problème de ce type de vérification est qu'il ne marche que si nous sommes dans le monde booléen. Lorsque nous voulons représenter des contraintes sur des valeurs, nous sommes obligés de donner une représentation abstraite du système (donc incomplète) sous forme booléenne. Par exemple, si tout un ensemble de code est conditionné par la condition "$x < 9$ and $x >= 9$" tel que x soit une variable entière, CLEM va introduire deux signaux booléens data1 et data2 représentant respectivement chaque contrainte élémentaire. Il conditionnera ensuite le groupe de codes associé par "data1 and data2". Or data1 et data2 sont opposés, donc l'expression data1 and data2 vaut toujours 0 : Tout le groupe de codes associé forme donc du code mort qui va quand même être généré par le compilateur CLEM... Ce code va également compliquer a posteriori la vérification de propriétés globales du système alors qu'il n'a aucune influence... (Voir 1.2)

C'est un des exemples de comportement dans CLEM qui pourrait trouver une interprétation différente grâce à l'implémentation de diagrammes de décisions linéaires.

La vision de ce stage est de contribuer à la conception d'un nouveau model checker inspiré de ceux utilisés par CLEM, qui s'appuierait sur une représentation sous forme de diagrammes de décisions linéaires. Cette représentation permettra d'identifier des intervalles de valeurs entières dans lesquelles une propriété est vérifiée.

Les diagrammes de décision linéaires étant des représentations permettant une interprétation abstraite du système, l'étude bibliographique est orientée dans ce sens. Nous nous sommes également appuyés sur une bibliothèque représentant un autre type de diagrammes de décisions linéaires développée en 2009 par Chaki, Gurfinkel et Strichman [CGS09] afin de faire apparaître certains principes de réduction dans les nouveaux diagrammes de décision linéaires qui sont identiques aux anciens.

Nous appellerons dans ce rapport LDD(s) les diagrammes de décisions linéaires selon [CGS09] et LDD$^+$(s) les diagrammes de décision linéaires sur lesquels nous avons travaillés.

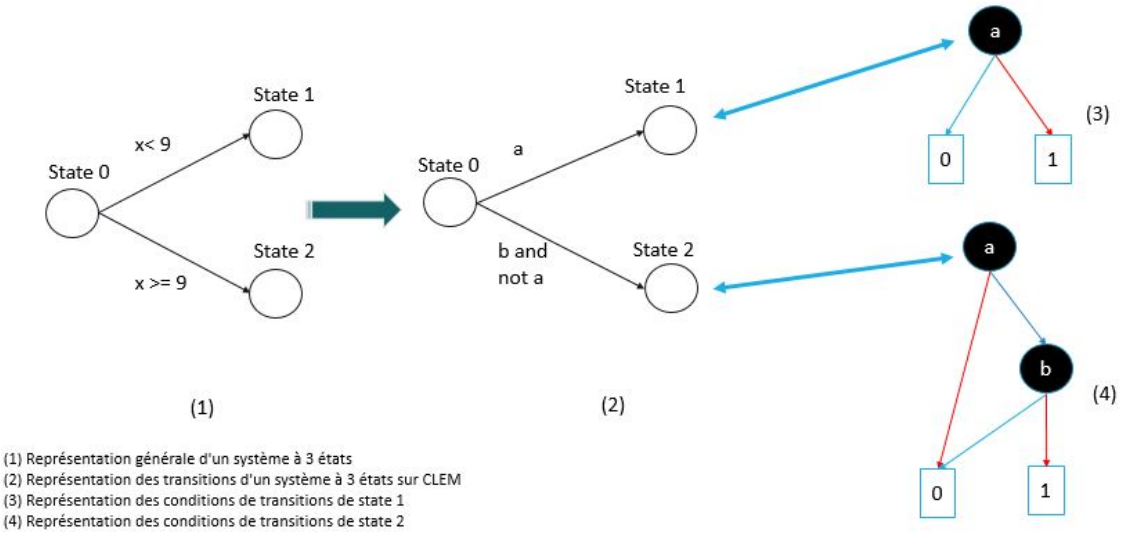

FIGURE 1.2: Exemple de fausse causalité dans CLEM en (1) et version acceptée par CLEM. "a" est le signal représentant x<9 et "b" le signal représentant x≤9 [Abd14]

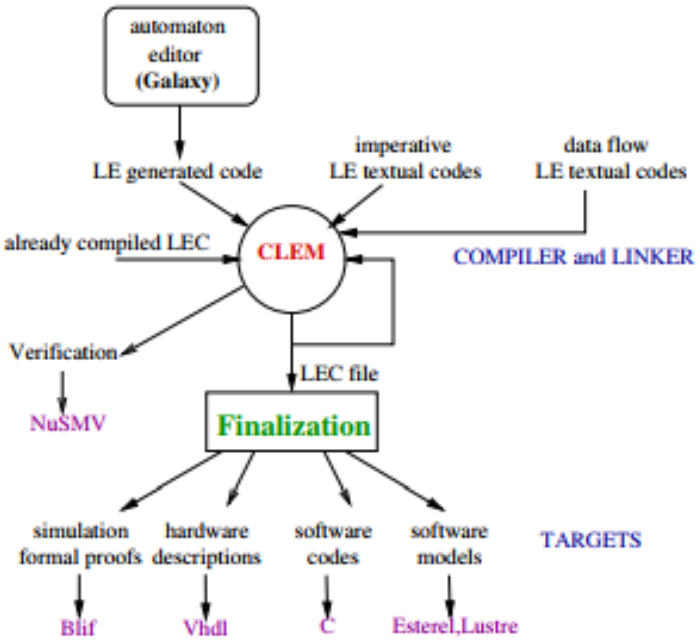

FIGURE 1.3: Schéma de compilation de CLEM

1.2 Présentation de l'organisme d'accueil : INRIA

L'Institut National de Recherche en Informatique et Automatique(INRIA) [INR], qui fete cette année ses 50 ans, est un établissement public français à caractère scientifique et technologique (EPST), qui a été créé à Rocquencourt en 1967, et qui s'est étendu sur huit sièges répartis sur tout le territoire français (Rocquencourt, Bordeaux, Grenoble, Lille, Nancy, Rennes, Saclay, et Sophia Antipolis).

INRIA est reconnu à l'échelle internationale, son but est de produire une recherche d'excellence en Mathématiques des Sciences du Numérique, et en Informatique et d'assurer son impact. Il a également pour vocation de faire du transfert technologique vers l'industrie en supportant financièrement le lancement de StartUps. Aujourd'hui, plus de 4000 personnes de différentes nationalités travaillent à l'INRIA, dont 3450 scientifiques (1678 chercheurs de l'institut et 1778 universitaires ou chercheurs d'autres organismes) qui sont regroupés en 172 équipes de recherche réparties dans les huit centres de recherche. Les thèmes de recherche de ces équipes s'articulent autour de cinq domaines principaux qui sont :

— Mathématiques appliquées, calcul et simulation
— Algorithmique, programmation, logiciels et architectures
— Réseaux, systèmes et services, calcul distribué
— Perception, Cognition, Interaction
— Santé, biologie et planète numérique

1.2.1 INRIA Sophia Antipolis

INRIA Sophia Antipolis est le troisième site créé par INRIA (après Rocquencourt et Rennes). Il est présent sur la technopole de Sophia Antipolis depuis 1981. Plus de 600 personnes travaillent dans ce centre de recherche, et au sein de 32 équipes (dont STARS). Ces équipes poursuivent un bon nombre d'engagement avec plusieurs acteurs académiques et économiques (des établissements de recherche, des associations, des entreprises ...), et s'impliquent dans des réseaux de recherche différents, en s'appuyant sur la qualité de ses chercheurs et ses services, afin d'ajouter de nouvelles idées innovantes qui enrichissent les domaines de recherche technologiques diversifiés tels que les mathématiques appliquées, les langages de programmation, l'algorithmique, les réseaux et systèmes distribués, etc.

1.2.2 Présentation de l'équipe projet STARS

L'équipe de recherche STARS (Spatio-Temporal Activity Recognition Systems) se concentre sur deux domaines d'application : la vidéo-surveillance et le maintien des personnes âgées à domicile. Elle se focalise sur la conception et le développement des systèmes cognitifs pour la reconnaissance d'activités. Cette équipe étudie les activités spatio-temporelles à long terme des êtres humains, des véhicules ou des animaux. Cette étude se fait à l'aide de l'interprétation sémantique et en temps réel de plusieurs scènes dynamiques surveillées par des capteurs et des caméras vidéo. STARS se focalise sur deux axes de recherche principaux qui sont :

— L'interprétation de scènes pour la reconnaissance d'activités : Cette interprétation a pour but de trouver une solution pour tout le problème d'interprétation, de l'analyse bas-niveau du signal jusqu'à la description sémantique du contenu de la scène contrôlée par les capteurs et/ou les caméras vidéo. Plus précisément, STARS travaille en perception, interprétation et apprentissage.

— L'architecture logicielle pour la reconnaissance d'activités : Ceci consiste à étudier les systèmes génériques pour la reconnaissance d'activités, et à élaborer des méthodologies de conception de ces systèmes, en assurant la généricité, la modularité, l'extensibilité, la ré-utilisabilité, la fiabilité et la maintenabilité.

1.3 Le projet NeuComp

L'avancée technologique ouvre des perspectives de recherche énorme dans le domaine des réseaux de neurones artificiels. Afin d'orienter ces recherches dans ce domaine en plein essor, l'Université Cote d'Azur a lancé l'académie d'excellence « Réseaux, Information et Société Numérique » (RISE).

Les objectifs principaux de l'académie RISE sont les suivants :

— L'élaboration et l'expérimentation de réseaux de communication du futur.
— L'excellence d'UCA dans les sciences numériques.
— Amélioration de la compréhension de la transformation émanant de la numérisation de la société.

Depuis le lancement de l'académie RISE, 8 projets ont déjà vu le jours :

1. Beyond RRTC
2. DigiCom
3. ElectroSmart
4. Digital Heart (LCN)
5. OPENING
6. UCA4SVR
7. ValueModels
8. **NeuComp : C'est dans le cadre de ce projet que se situe mon travail.**

Dans le cadre de NeuComp, 3 axes principaux de recherche ont été mis en place dans le but de concevoir un modèle de réseau de neurone à Spike s'inspirant du modèle *Leaky Integrate and Fire* :

i. **La modélisation d'un neurone à Spike :** Ce travail effectué au sein du I3S consiste à modéliser et simuler de manière discrète et temporelle différents réseaux de neurones utilisant le modèle *Leaky Integrate and Fire [LIF]* sur une architecture neurmorphique multi-coeur dédiée (SpiNNaker).

Responsables : A. Muzy et E. Demaria.

Stagiaires : J. Goutey et T. Lyvonnet.

ii. **La vérification des propriétés temporelles d'un réseau de neurone :** Ce travail effectué au sein de l'INRIA de l'I3S et du LEAT consiste à modéliser les performances de différents réseaux, implémenter des simulations dans un langage dédié au Model Checking afin d'extraire les propriétés temporelles de ce type d'archétype. C'est plus précisément dans ce cadre que s'est déroulé mon stage.

Responsables : A. Ressouche, D. Gaffé et E. Demaria.

Stagiaires : C. Girard, D. Havyarimana.

iii. **L'implémentation hardware d'un réseau de neurone :** Ce travail effectué au sein du LEAT consite à concevoir l'architecture neuromorphique d'un réseau de neurones à Spike

s'inspirant du modèle *Leaky Integrate and Fire* en se focalisant sur les gains en surface et en consommation d'énergie, et ce comparé aux modèles neuronaux classiques de type perceptrons multicouches.

Responsable : B. Mirammond.

Stagiaires : L. Khacef et U. Alakbarova.

Chapitre 2

État de l'art

Dans le contexte du projet expliqué ci-dessus, il est important de faire une étude des recherches scientifiques concernant la vérification formelle et l'interprétation abstraite en général, ainsi que les différents types de représentation abstraite utilisés à ce jour afin de comprendre le pourquoi d'un tel travail, et son évolution potentielle vers un type de model checker basé sur les LDDs^{+}.

2.1 Les bugs

Le terme **«bug»** est utilisé pour désigner toute sorte de panne dans le fonctionnement d'un système informatique résultant d'une erreur lors de la conception du logiciel informatique. Ces pannes sont généralement dues à deux choses :

- Le dysfonctionnement d'un appareil ;
- Le dysfonctionnement d'un programme.

L'exemple présenté en figure 2.1, nous donne un exemple de bug dans le programme de calcul du factoriel. Cette erreur est due au fait que la fonction fact(n) ne coïncide avec n ! que pour $1 \leq n \leq 12$, vous trouverez ainsi une démonstration en annexe que 34 ! vaut 0 sur 32 bits.

La gravité du dysfonctionnement peut être :

- **Bénigne** : ce cas se résume la plupart du temps à la défaillance d'un système logiciel, une perte d'informations, etc.
  Ex : Blocage ou erreur des systèmes de réservation en ligne ou sites Web, ATM, PS3, ordinateurs, téléphones, etc.
- **Majeure** : il peut entraîner de graves conséquences
  Ex : La Lexus ES350 a tué un grand nombre de ses propriétaires en enclenchant l'accélération jusqu'à 150km/h tout en désactivant la pédale de frein à cause de dysfonctionnement de son ordinateur de bord, l'explosion d'Ariane 501 en 1996[La96], perte de satellites, Bug dans la division flottante du Pentium, calcul non précis depuis le lancement dû à un cumul d'erreurs arithmétiques d'arrondis du missile Patriot en 1991[Arn91], etc.

Il faut savoir que plus un code est long, plus il est difficile de localiser un bug. Il existe des bugs qui résultent de combinaison de conditions imprévues et improbables. Ce genre de bug nécessite des outils de vérification performants tel que la vérification formelle par Model Checking ou par l'interprétation abstraite, etc.

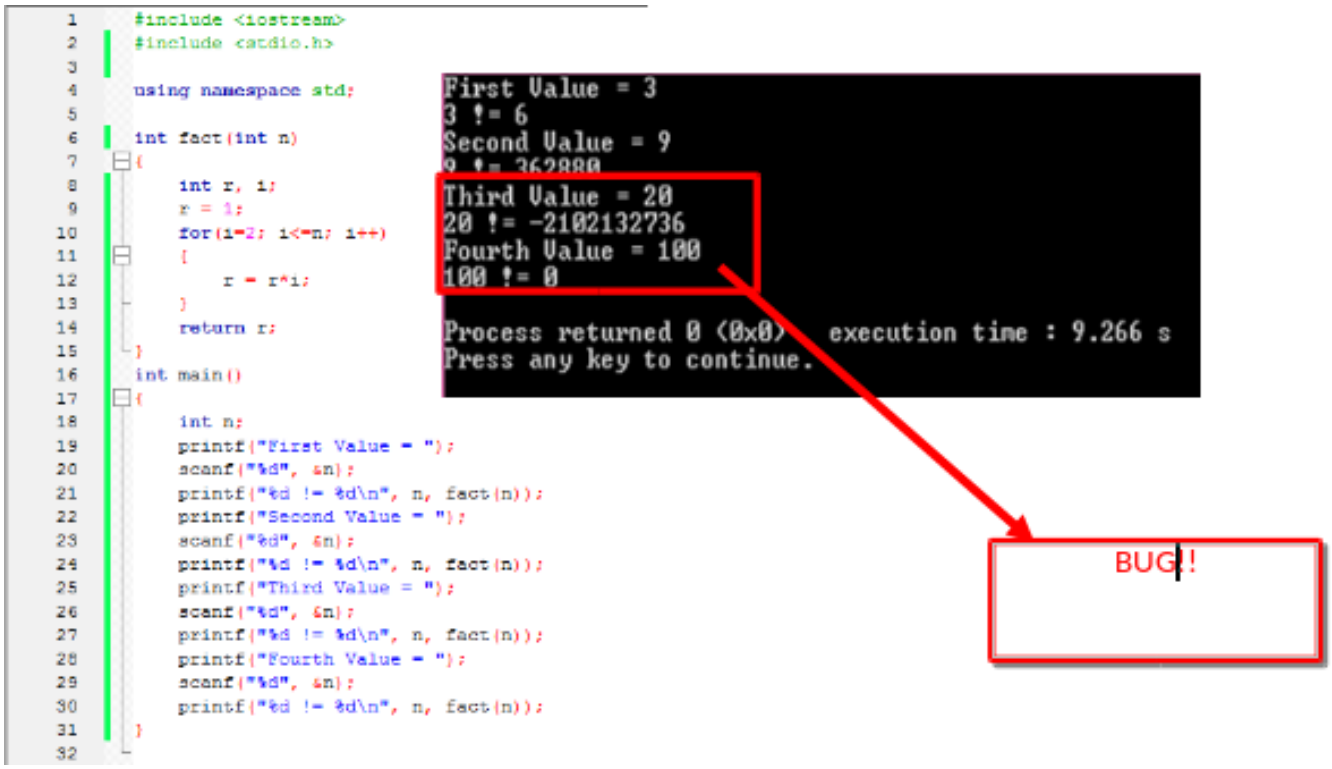

FIGURE 2.1: Exemple d'un bug du programme factoriel, vous trouverez en annexe une démonstration de l'origine de ce bug.

2.2 Le Model Checking

Le Model Checking ou la vérification de modèle est une technique destinée à vérifier les propriétés des systèmes hardware ou software dans les cas où toute erreur de fonctionnement du système serait inacceptable. Le modèle est un graphe orienté : un sommet représente un état du système et chaque arc représente une transition, c'est-à-dire une évolution possible du système d'un état donné vers un autre état. Un tel graphe permet de construire une structure de Kripke[Kri63] qui la représentation pratique du modèle dans les model checkers.

Schématiquement, un algorithme de Model Checking prend en entrée un modèle du programme, une propriété exprimée dans la logique temporelle et répond si le comportement satisfait ou non la propriété. Le Model Checking a pour avantage de fournir un contre exemple quand une propriété du système n'est pas vérifiée, cette caractéristique lui a valu un grand succès dans le milieu industriel.

Le principe du Model Checking est le suivant :

«Étant donnée une structure de Kripke M, un état s appartenant à S et une formule logique temporelle f, le Model Checking consiste à établir si M, s $\models$ *f.»*

L'algorithme décide si la formule f est vraie dans l'état s du système.

Les propriétés qui peuvent être vérifiées en utilisant un model checker sont d'une nature qualitative : le résultat généré est-il correct ? Le système peut-il atteindre une situation de blocage ? On peut soit vérifier que l'on peut atteindre un état où la formule est fausse depuis l'état initial, (le moyen utilisé pour cela est une exploration en avant en partant de l'état initial jusqu'à atteindre un état prédéfini), soit l'analyse en arrière c'est à dire partir d'un état établi et vérifier si on peut atteindre l'état initial.

Par exemple, quand deux programmes concurrents s'attendent mutuellement, peuvent-ils arrêter le système entier ? Des propriétés peuvent être vérifiées : peut-il se produire un blocage du système après une heure de réinitialisation ? Ou, une réponse peut-elle toujours être reçue après un certain temps ? Le Model Checking demande une déclaration précise des propriétés qui doivent être examinées car en faisant un modèle formel d'un système, cette étape conduit souvent à la découverte de plusieurs d'incohérences dans la documentation. Le modèle du système est habituellement généré automatiquement par la sémantique du langage de description hardware.

La vérification formelle repose sur 3 éléments :

— La représentation mathématique du système à vérifier M.
— Le langage formel qui exprime la propriété μ du système.
— Un algorithme de vérification du bon (ou mauvais) fonctionnement du système.

La spécification du système prescrit ce que le système doit faire, et non ce qu'il ne doit pas faire ; tandis que la description du modèle aborde la façon dont le système doit se comporter. Le model checker examine tous les états significatif du système pour vérifier s'il satisfait la propriété désirée. Si nous rencontrons un état dont la propriété n'est pas vérifiée, le model checker fournit un contre-exemple indiquant à l'utilisateur le moyen d'atteindre l'état indésirable. Le contre-exemple décrit un chemin d'exécution conduisant de l'état initial du système à l'état où la propriété n'est pas vérifiée.

À l'aide d'un simulateur, l'utilisateur peut reproduire le scénario qui a généré l'erreur ; nous obtenons, de cette manière, des informations utiles de correction et il est alors possible d'adapter le modèle (ou la propriété) en tant que tel.

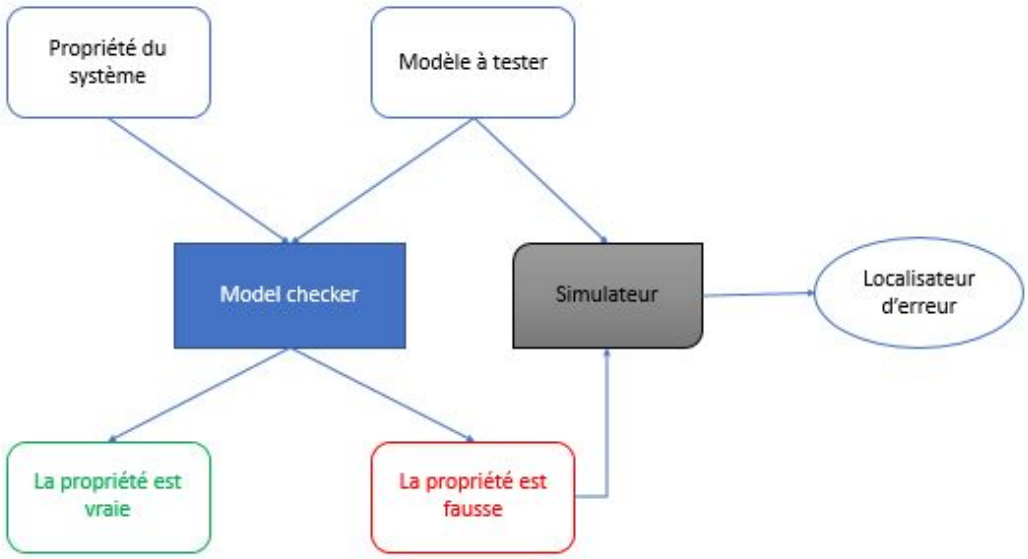

FIGURE 2.2: Approche d'un model checker

Avantage du Model Checking

Il y a 2 avantages à utiliser le Model Checking :

- le premier est qu'il est complètement automatique.
  Nous aurons en entrée S, le système à vérifier, μ la propriété à tester et en sortie nous avons soit **Oui** quand la propriété est vérifiée et **Non** dans le cas contraire.
- le second est qu'il peut fournir un contre exemple
  En effet, dans le cas ou la propriété à tester n'est pas vérifiée le Model Checker est capable de nous fournir un contre exemple ce qui nous permet de comprendre le comportement du système même dans le cas ou il n'a pas le comportement escompté.

Inconvénient du Model Checking

- Il souffre du problème de l'explosion combinatoire (explosion d'espace d'états).
  Le nombre d'états nécessaires pour modéliser le système avec précision peut facilement dépasser la quantité de mémoire disponible. Malgré le développement de plusieurs méthodes très efficaces pour lutter contre ce problème, les modèles de systèmes réalistes peuvent encore être trop grands pour tenir en mémoire.
- Il n'est pas garanti pour donner des résultats corrects.
  Comme tout outil, un vérificateur de modèle peut comporter des défauts logiciels.
- La limite d'expression des propriétés dans la logique temporelle.

2.3 L'interprétation abstraite

L'interprétation abstraite est une théorie de l'approximation basée sur les travaux réalisés initialement par Patrick Cousot [PR76]. Son but étant d'utiliser au mieux la notion d'abstraction en constituant une zone géométrique qui couvre toutes les exécutions pour donner un modèle abstrait des systèmes complexes sous la forme d'intervalle, ellipse, octogone ou polygones au lieu de nombres individuels. Le domaine d'abstraction est déterminé par le domaine de la propriété à tester ; pour cela, on détermine des sémantiques liées par des relations d'abstraction où la caractéristique fondamentale est que la plupart des propriétés dépendent de la notion d'abstraction.

Par le biais de cette représentation abstraite, on peut vérifier la robustesse et/ou respect de propriété où les zones ou la propriété n'est pas vérifiée sont considérées comme des zones interdites, il faut donc que dans le test, l'intersection de la zone représentant tout les chemins possible d'un système et les zones interdites soit nulles(Voir figure 2.3).

Certaines réponses concrètes ne peuvent trouver que des réponses approximatives par la sémantique abstraite qui sont des représentations mathématiques de tous les comportements possibles d'un programme quelles que soient ses entrées ; cependant, il est possible de traiter des problèmes infinis en réalisant une abstraction finie de ceux-ci d'où sa grande utilité pour traiter des problèmes industriels.

Malgré les réponses incomplètes de cette interprétation abstraite, elle permet tout de même au compilateur de répondre à des questions qui n'ont pas besoin d'une connaissance approfondie du système ou qui tolèrent une réponse correcte mais incomplète. En tenant compte des propriétés de vivacité et de sûreté(c'est à dire qu'on ne va pas dans des zones interdites), on représente les traces d'exécution d'état de plusieurs points caractérisant un système suivant un pas de calcul.

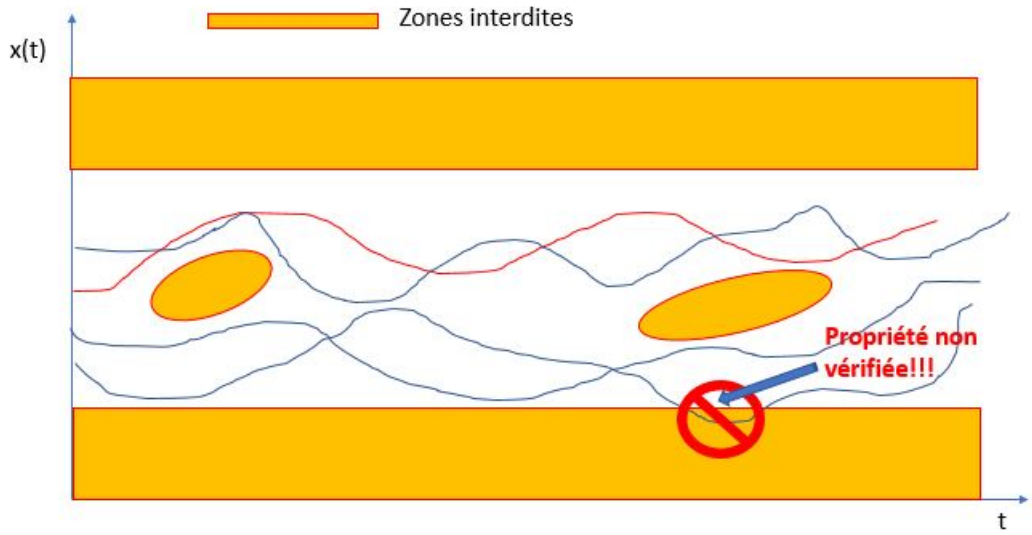

FIGURE 2.3: Représentation abstraite selon Cousot [Cou78]

2.3.1 Les types de représentation abstraite

Vous trouverez un récapitulatif des différents types d'abstraction étudiées à ce jours en figure 2.4 où vous pourrez y voir les polyèdres qui sont ceux qui représentent le mieux nos LDDs$^+$.

2.3.2 Les polyèdres

Selon le dictionnaire, on définit un polyèdre comme :

"un solide délimité par des faces polygonales dont les intersections forment des arêtes et les points de rencontre de celles-ci, des sommets".

En mathématique, le terme "polyèdre" est utilisé pour désigner toute sorte de constructions reliées, qu'elles soient géométrique que purement algébriques ou abstraites.

On distingue les polygones convexes et les polygones non convexes. Les polyèdres convexes étant les polyèdres possédant ses diagonales contenues dans son intérieur ; lorsqu'un polyèdre est convexe et borné, il est appelé polytope. Les polyèdres non convexes sont des polyèdres qui possèdent au moins une face sur laquelle il ne peuvent pas être posés.

Les polyèdres sont utilisés pour représenter le domaine dans lequel une propriété est vérifiée et/ou le domaine ou une propriété ne devrait pas l'être (la zone interdite). Dans le chapitre suivant, vous trouverez une présentation de diagrammes de décisions linéaires outil de représentation abstraite d'un système

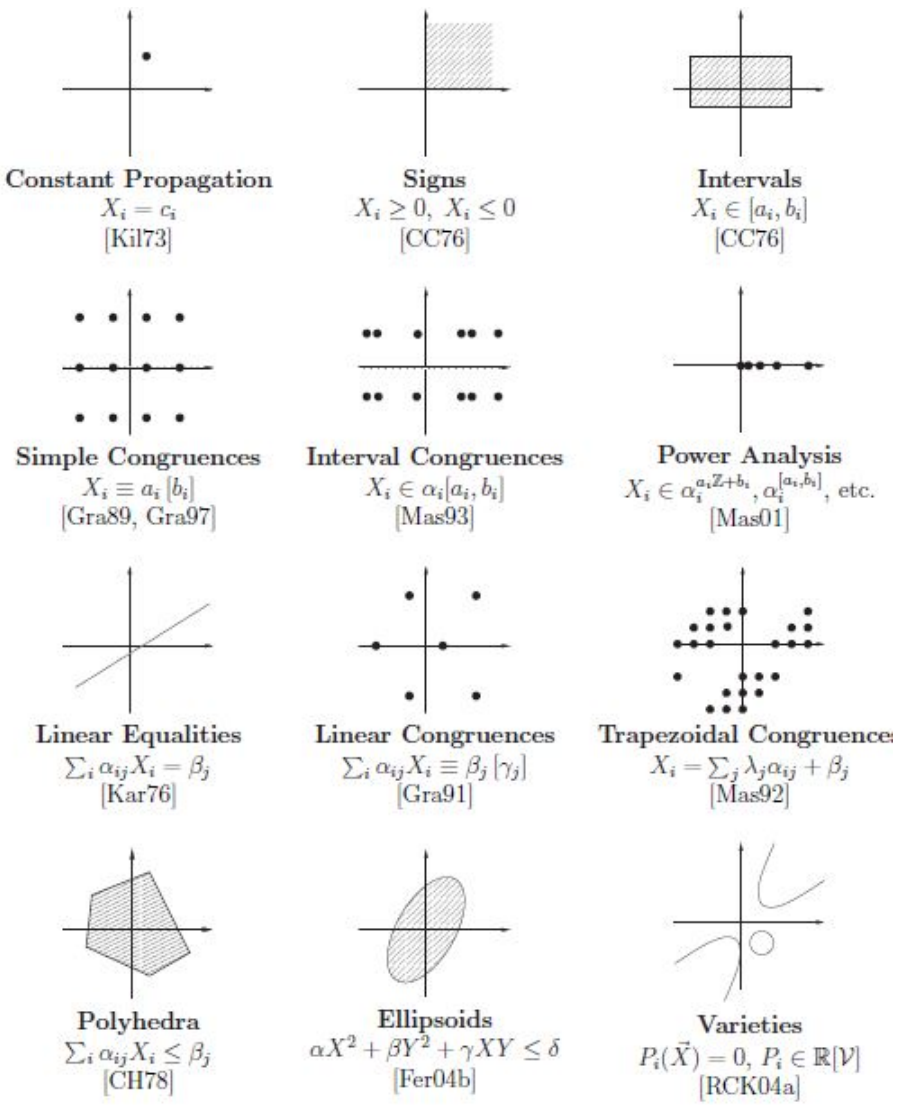

FIGURE 2.4: Type de représentation abstraite extrait de la thèse de Antoine Miné [Min04]

Chapitre 3

Les diagrammes de décisions linéaires

3.1 Introduction

Nous allons dans ce chapitre vous présenter une représentation de diagramme de décision s'appuyant sur des inéquations linéaires mais qui diffère des LDDs d'origine de par leur format. Les LDDs$^+$ développés actuellement sont inspirés du modèle de 2009 [CGS09] où leurs concepteurs nous présentent des diagrammes de décision inspirés par les DDDs[MLAH99] qui évitent de décrire le système au niveau booléen mais sous forme d'inéquations linéaires.

Nous distinguons les LDDs des LDDs$^+$ par le format de leurs inéquations. En effet, nous travaillons sur des diagrammes de décisions supportant une grande marge de contraintes sous la forme :

Pour tout $a \in \mathbb{Z}^+$ et c $\in \mathbb{Z}$

$$\sum_{i=0}^{n} a_i x_i \leq c$$ [1]

Les LDDs[CGS09] sont des outils de représentation abstraite qui utilisent des diagrammes de décision binaires avec des nœuds représentés sous forme d'inéquations linéaires satisfaisant une théorie d'ordonnancement et des contraintes de réduction locale. Ces LDDs sont implémentés au-dessus de CUDD qui est le paquet de manipulation des BDDs développé par Somenzi à partir de la théorie UTVPI. La théorie UTVPI (Unit Two Variable Per Inequality) est une représentation des contraintes sous la forme :

Pour tout $a \in$ {-1,0,1} et c $\in \mathbb{Z}$

$$\sum_{i=0}^{n} a_i x_i \leq c$$

La bibliothèque LDD$^+$ est implémentée au dessus des diagrammes de décision binaires à arcs complémentés afin de garantir une meilleure optimisation. Ces LDDs$^+$ ont une représentation symbolique sous forme de polyèdre qui reste identique aux LDDs mais possèdent un pouvoir de représentation plus important. En effet, on peut voir qu'il différent de part le degré de liberté des coefficients a_i, a_i appartenant à {-1,0,1} pour les LDDs et appartenant à $\mathbb{Z}^+$ pour les LDDs$^+$.

Les LDDs ont permit une manipulation symbolique des données numériques[Abd14] en usant d'inéquations du type $\sum_{i=0}^{n} a_i x_i \leq c$, $\sum_{i=0}^{n} a_i x_i < c$, $\sum_{i=0}^{n} a_i x_i \geq c$, $\sum_{i=0}^{n} a_i x_i > c$ en intégrant différentes opérations sur les inéquations tels que :

1. Les algorithmes présentés en 1 et 2 considère a_i comme $coeff_i(-)$ et c comme $limit(-)$

— L'intersection ou l'union de deux inéquations.
Par exemple, x $<$ a $\cup$ x $\geq$ a représente l'ensemble $\mathbb{Z}$ tout comme x $<$ a $\cap$ x $\geq$ a représente l'ensemble vide $\varnothing$
— L'opérateur d'égalité se retrouve déduit de x $\leq$ a $\cap$ x $\geq$ a
— Une simplification des expressions beaucoup plus poussée est possible pour des expressions comme x-y $\leq$ 3 $\cap$ x-t $\geq$ 8 $\cap$ y-z $\leq$ 6 qui se réduisent jusqu'à être représenté par l'expression x-z $\leq$ 9 $\cap$ x-t $\geq$ 8. L'opérateur de simplification intègre également des simplifications en cas d'implication faible (ex : x$\leq$3 $\cup$ x$\leq$8) ainsi que les cas d'implication forte (ex : x$\leq$3 $\cap$ x$\leq$8) ces deux cas sont schématisés en figure 3.5

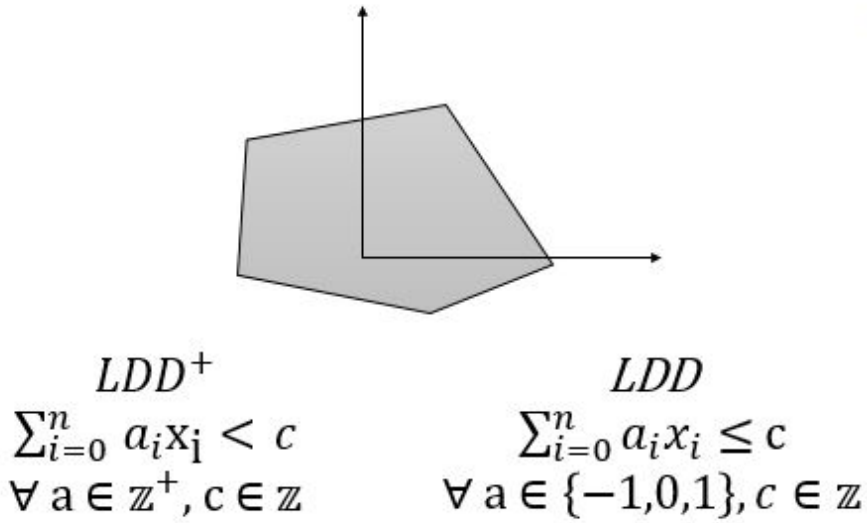

FIGURE 3.1: Représentation d'un LDD sous forme de polyèdre

3.2 Structure des LDDs$^+$

Que ce soit les LDDs ou les LDDs$^+$, ils sont tout les deux définis par un graphe acyclique orienté ayant la même structure qu'un BDD[Bry86] présenté par :

— Deux nœuds terminaux étiquetés respectivement par 0 et 1 ;
— Des nœuds non terminaux. Chaque nœud non terminal "u" est étiqueté par un prédicat ou un terme atomique noté C(u) et possède deux descendants désignés par *Then*(u) et *Else*(u) ;
— Des arêtes (u, *Then*(u)) et (u, *Else*(u)) pour chaque nœud non terminal "u".

Voici un exemple de représentation LDD de deux suites d'inéquations F1 et F2. F1 est simple, convexe et possède 3 inéquations tandis que F2 est non convexe, beaucoup plus complexe et possède 6 inéquations.

*F1 = x + y $\geq$ 25 and 7*x + 3*y $\leq$ 163 and x - y $\geq$ 9* (Voir figure 3.2).

*F2 = 4*x + 5*y $\geq$ 45 and not (-3*x + 7*y $<$ -13) and not (- x -2*y $<$ -26) and not (- x - y $<$ -19) and not (2*x -3*y $<$ -17)* (Voir figure 3.3).

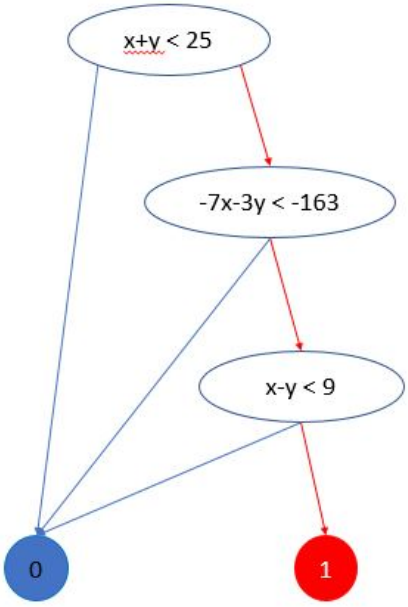

FIGURE 3.2: Représentation de F1 sous forme de LDD$^+$ avec $n = 3$

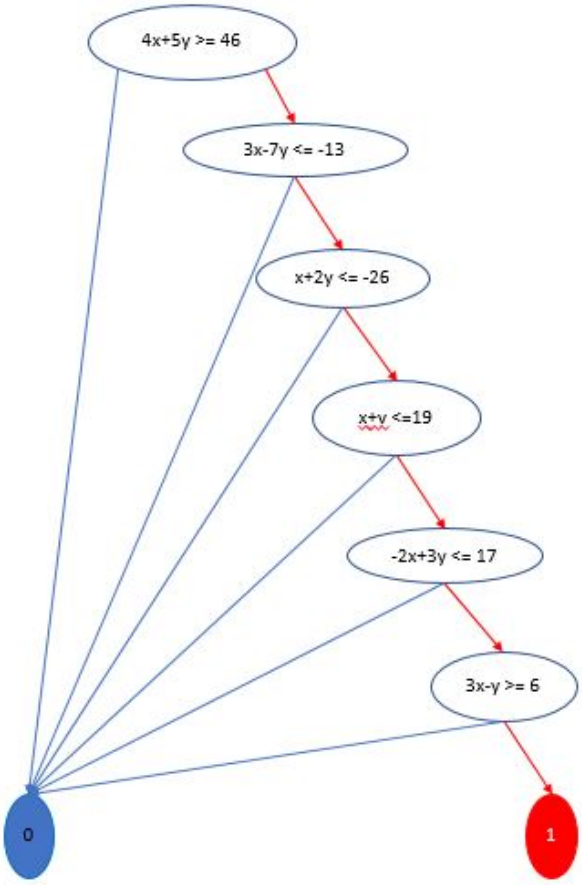

FIGURE 3.3: Représentation de F2 sous forme de LDDs$^+$ avec $n = 6$

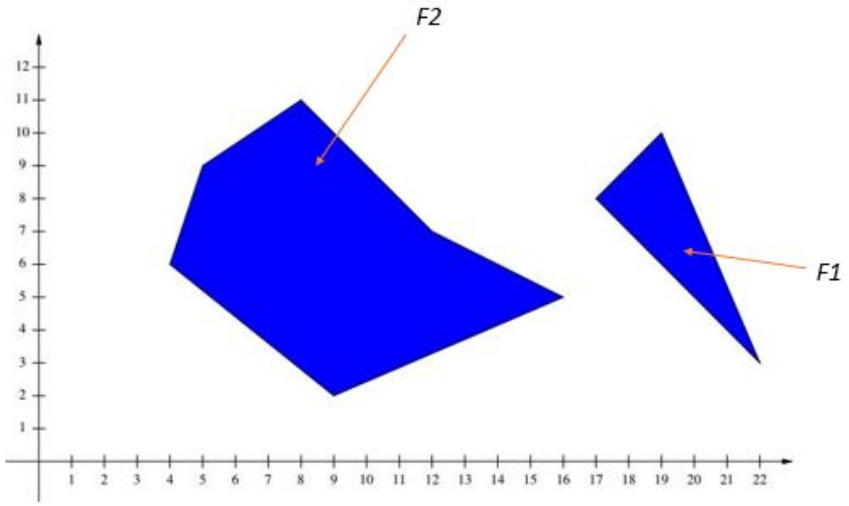

FIGURE 3.4: Représentation de F1 et F2 dans le domaine abstrait(le nombre d'arêtes correspond à la valeur de n)

3.3 Les règles de réduction des LDDs$^+$

Puisque les LDDs nous permettent une analyse des données numériques tout au long de notre parcours de graphe, le principe de l'inclusion naissant de ces LDDs nous permet de réduire l'état des transitions tels que présenté en figure 3.5.

Un LDD est réduit localement si et seulement si les cinq conditions suivantes sont satisfaites sur chaque nœud interne u et v :

1. Pas de nœuds dupliqués : attr(u) = attr(v) <=> u = v
2. Pas de nœuds redondants : $Then$(v) $\neq$ $Else$(v).
3. Les étiquettes sont normalisées : C(v) = N(C(v)).
4. Implication forte : ¬ IMP(C(v),C($Then$(v))). Voir 3.5 (1).
5. Implication faible. IMP(C(v),C($Else$(v)))) $Then$(v) $\neq$ $Then$($Else$(v)). Voir 3.5 (3).

Les points 1,2 et 3 sont identiques au conditions de réduction des BDDs, en figure 3.5 une représentation de cas de réduction est traités, il s'agit des cas d'implication forte en 4, faible en 5. La figure 3.6 traite d'un cas qui englobe à la fois les cas d'implications fortes et faibles. Ces 3 cas constituent la problématique à traiter durant le stage.

Comment appliquer ces réductions propres au LDD sur les LDDs$^+$?

3.3.1 Réduction par implication forte ou faible

Avec les LDDs$^+$, nous pouvons remarquer que nous utilisons la notions de domaines, ces domaines nous permettent de diminuer les nœuds a tester dans les cas où il existe des nœuds LDD représentant un domaine inclut dans d'autres nœuds.

Par exemple, en figure 3.5, pour le cas de l'implication forte, lorsque nous avons confirmé que x est inférieur à 3, le cas où x serait supérieur à 8 est déjà exclut et étant certain que x est inférieur à 8, cette représentation nécessite une réduction du nœud représentant x<8 qui ne donne aucune information supplémentaire. Il en est de même pour l'implication faible ou le

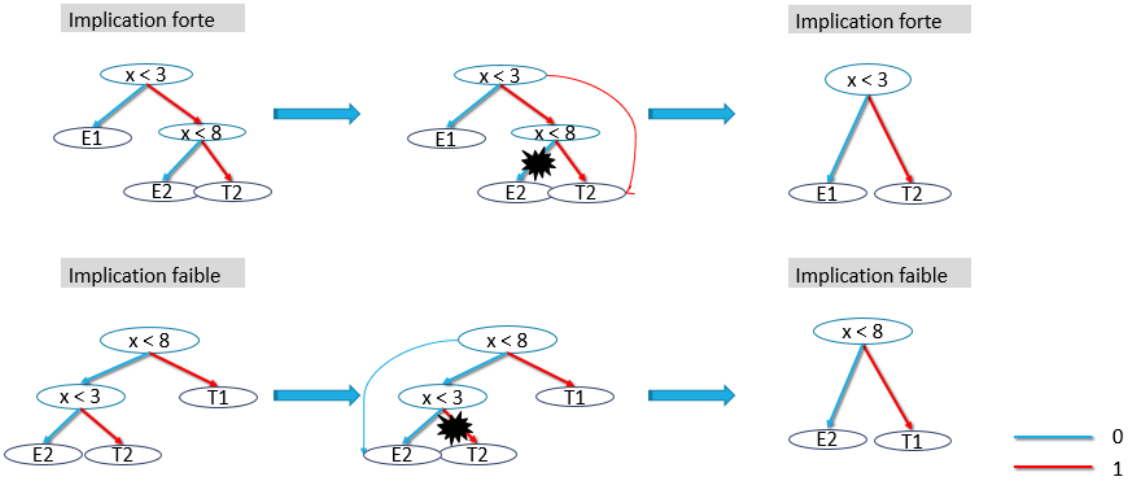

FIGURE 3.5: Réduction des LDDs : Cas des implications fortes (1) et faibles (3)

test du nœud x<3 n'apporte pas d'information supplémentaire il sera donc supprimé puisque x est déjà supérieur ou égal à 8 ; il faut réduire ces deux cas selon des algorithmes bien définis en 1 et 2. Ces algorithmes s'arrêtent lorsqu'ils tombent sur une feuille (LDDNODE1).

Algorithm 1 implyhigh(LDDnode u)

```
1:  if(u != LDDNODE1)
2:    LDDnode v ← Then(u)
3:    LDDnode w ← Else(u)
4:    implyhigh(w)
5:    if(v != LDDNODE1)          ▷ LDDNODE1 : feuille 1 correspondant à l'absence de
      contraintes
6:      if (∀ i coeff_i(u) = coeff_i(v) and limit(u) < limit(v))
7:        Then(u) ← Then(v)
8:        Free(v)
9:        v ← Then(u)
10:     endif
11:     implyhigh(v)
12:   endif
13: endif
```

3.3.2 Inclusion de nouvelles combinaisons dans les LDDs$^+$

Nous arrivons maintenant au cœur du travail effectué durant le stage, au regard de la problématique citée ci-haut, mon travail personnel a consisté à inclure automatiquement et au bon endroit, de nouvelles variables BDD correspondant chacune à une combinaison linéaire d'inéquations dans les LDDs$^+$ car nous avons observé, dans les LDDs, qu'il s'opérait un affinement du polyèdre en passant de $F= x < 8$ *and* $y < 3$ *and* $x+y < 16$ à $F= x < 8$ *and* $y < 3$ *and* $x+y < 11$, ce raffinement étant très avantageux au LDDs$^+$ nous avons voulu les incorporer. Le LDD affine ses inéquations en rajoutant des nœuds issus de recombinaison de deux nœuds du graphe, il regarde si le nouveau nœud va lui servir à quelque chose, si oui il le garde si non il ne l'utilise pas.

Algorithm 2 implylow(LDDnode u)

```
1: if(u != LDDNODE1)
2:   LDDnode v ← Else(u)
3:   LDDnode w ← Then(u)
4:   implylow(w)
5:   if(v != LDDNODE1)
6:     if (∀ i coeff_i(u) = coeff_i(v) and limit(u) > limit(v))
7:       int toggle ← toggle(u) * toggle(v)
8:       Else(u) ← Else(v)
9:       toggle(u) ← toogle
10:      Free(v)
11:      v ← Else(u)
12:    endif
13:    implylow(v)
14:  endif
15: endif
```

Algorithm 3 Réduction en cas de recombinaison par addition et/ou soustraction (exemple du cas 3.8)

```
1: if(u != LDDNODE1)
2:   LDDnode v ← Then(u)
3:   LDDnode w ← Else(v)
4:   if(v != LDDNODE1)
5:    LDDnode x = Then(v)
6:    if(x != LDDNODE1)
7:    if (∀ i coeff_i(u) + coeff_i(v) = coeff_i(w) and limit_i(u) + limit_i(v) < limit_i(w))
8:      LDDnode y ← NewNodePlus(u,v)
9:     endif
10:      Then(x) ← y
11:      Else(y) ← x
12:      Then(y) ← Then(x)
13:    endif
14:   endif
15:  endif
```

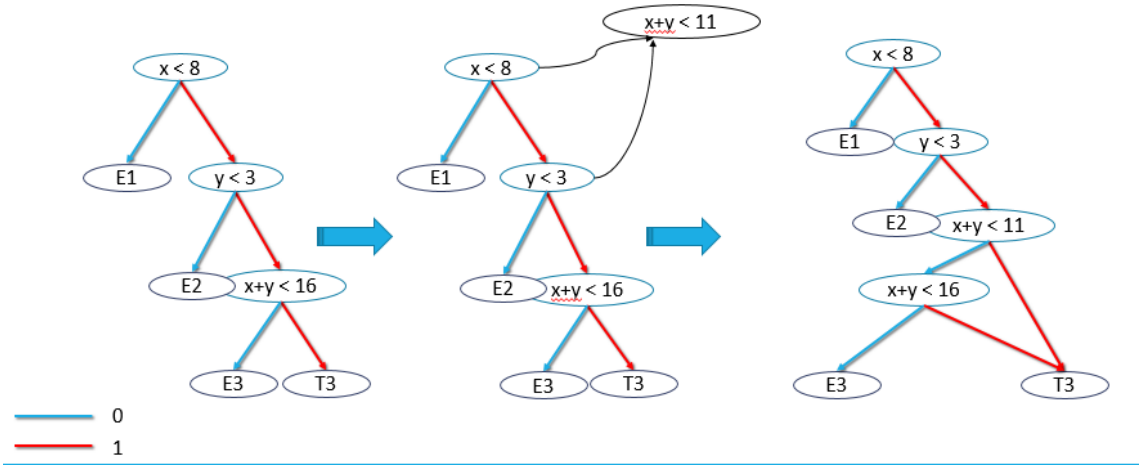

FIGURE 3.6: Inclusions des combinaisons

Nous avons appliqué ce même principe pour F1 et F2, des LDDs généraux et il se trouvent que ces recombinaisons sont très avantageuses car nous obtenons les bornes de chaque inéquation utile pour la génération d'un polyèdre affiné.

La liste des inéquations initiales en F1 est présentée en table 3.1, celle de F2 en table 3.2. Avec le principe de la recombinaison nous obtenons de nouvelles inéquations dans l'arbre LDD pour les deux cas.

TABLE 3.1: Liste des contraintes LDDs de F1

$Table[n]$	**Inéquations**
Table[0]	x+y < 25
Table[1]	-7x-3y < -163
Table[2]	x-y < 9

TABLE 3.2: Liste des contraintes LDDs de F2

$Table[n]$	**Inéquations**
Table[0]	4x+5y < 45
Table[1]	-3x + 7y < -13
Table[2]	-x − 2y < -26
Table[3]	-x - y < -19
Table[4]	2x-3y < -17
Table[5]	3x-y < 7

TABLE 3.3: Recombinaison des contraintes LDDs de F1

$Table[n]$	**Noeud 1**	**Noeud 2**	**Addition**	**Soustraction**
Table[0]	x+y < 25	-7x-3y < -163	-x < -22, y < 3	—
Table[1]	x+y < 25	x-y < 9	x < 17	y < 8
Table[2]	-7x-3y < -163	x-y < 9	-y < -10	-x < -19

TABLE 3.4: Recombinaison des contraintes LDDs de F2

Table[n]	**Noeud 1**	**Noeud 2**	**Addition**	**Soustraction**
Table[0]	4x+5y < 45	-3x + 7y < -13	43y < 83	43x < 380
Table[1]	4x+5y < 45	-x − 2y < -26	-3y < -59, 3x < -40	—
Table[2]	4x+5y < 45	-x - y < -19	3x < -40, y < -31	—
Table[3]	4x+5y < 45	2x-3y < -17	-x < -50	11y < 79
Table[4]	4x+5y < 45	3x-y < 7	11x < 25	19y < 107
Table[5]	-3x + 7y < -13	-x − 2y < -26	19x < 80	y < 5
Table[6]	-3x + 7y < -13	-x - y < -19	-x < -16	5y < 22
Table[7]	-3x + 7y < -13	2x-3y < -17	5y < -73, 5x < -158	—
Table[8]	-3x + 7y < -13	3x-y < 7	y < -1, x < 2	—
Table[9]	-x − 2y < -26	-x - y < -19	—	-y < -7, x < 12
Table[10]	-x − 2y < -26	2x-3y < -17	7y < -69	7x < -44
Table[11]	-x − 2y < -26	3x-y < 7	-7y < -71	-7x < -40
Table[12]	-x - y < -19	2x-3y < -17	-y < -11	-x < -8
Table[13]	-x - y < -19	3x-y < 7	-2y < -25	-2x < -13, -7y < -65
Table[14]	2x-3y < -17	3x-y < 7	—	-7x < -38

Analyse des résultats :

Grâce à ces recombinaisons, nous avons pu extraire de nouvelles inéquations qui nous permettent d'encore plus affiner la représentation du système. Il reste à construire le LDD$^+$ qui contiendra les nouvelles inéquations et les anciennes, pour cela il faut établir une liste de dépendance des variables où on trouve les informations concernant l'origine de chaque nœud LDD$^+$; aussi pour une représentation ordonnée les nœuds LDDs$^+$ doivent être placée de telle sorte que les nœuds contenant les mêmes variables se suivent afin d'étudier leurs éventuelles réductions par implication faible/forte. Étant donnée que, pour des systèmes complexes, la taille du LDD$^+$ serait trop grande pour prendre en compte toutes ses inéquations, il faut établir un système de sélection des nœuds les plus intéressants (représentatifs) afin d'incorporer le moins de nœuds supplémentaires possibles.

Généralisation des résultats

Nous avons généralisé la méthodologie de branchement des nouveaux nœuds aux pré-existants telle que représentée dans l'exemple suivant représenté dans les figures 3.7 3.8 3.9 3.11 :

"Soit C1 : x<3 et C2 : y<-1, ayant des recombinaisons égales à x+y < 2 : C1+C2 et x-y < 4 : C1-C2, soit les zones 1, 2, 3, 4 les possibles domaines de représentation du système."

Si dans l'arbre LDD, il existe des noeuds identiques en terme de coefficients aux noeuds obtenus par recombinaisons et qui diffère uniquement par la limite, il faut en fonction de chaque cas introduire de nouveau branchement et/ou noeuds. Les cas extraits sont représentés en 3.8, 3.9, 3.10, 3.11 ainsi qu'une représentation générale des 4 cas possible dans un seul arbre LDD en 3.12.

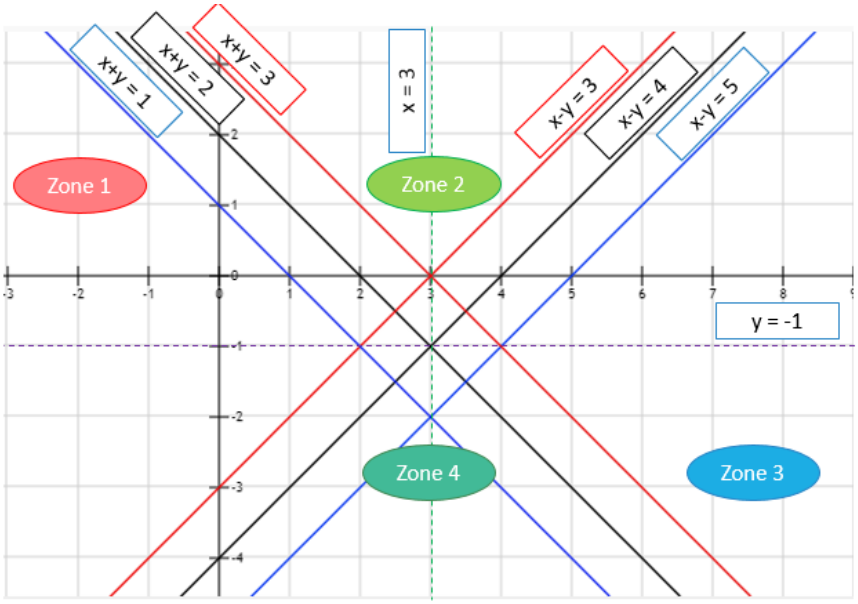

FIGURE 3.7: Représentation du LDD à l'aide des zones

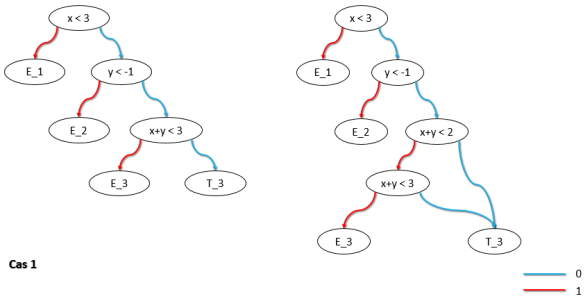

FIGURE 3.8: Premier cas

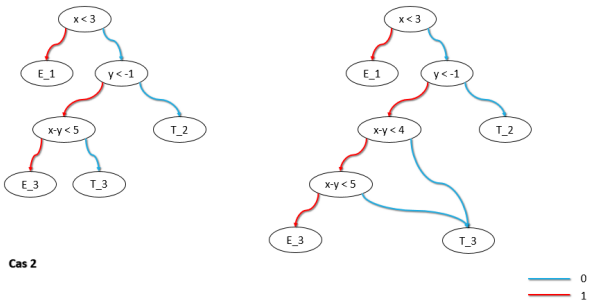

FIGURE 3.9: Deuxième cas

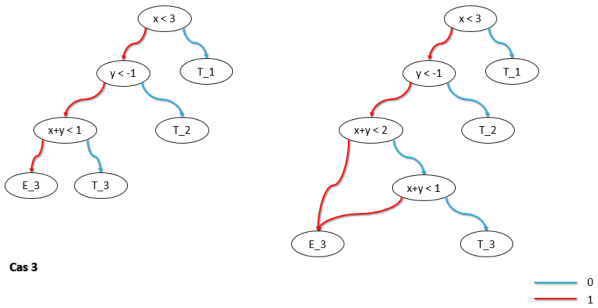

FIGURE 3.10: Troisième cas

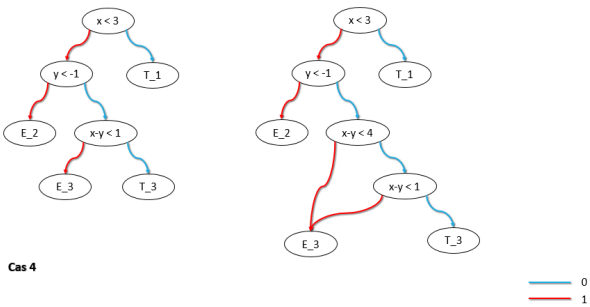

FIGURE 3.11: Quatrième cas

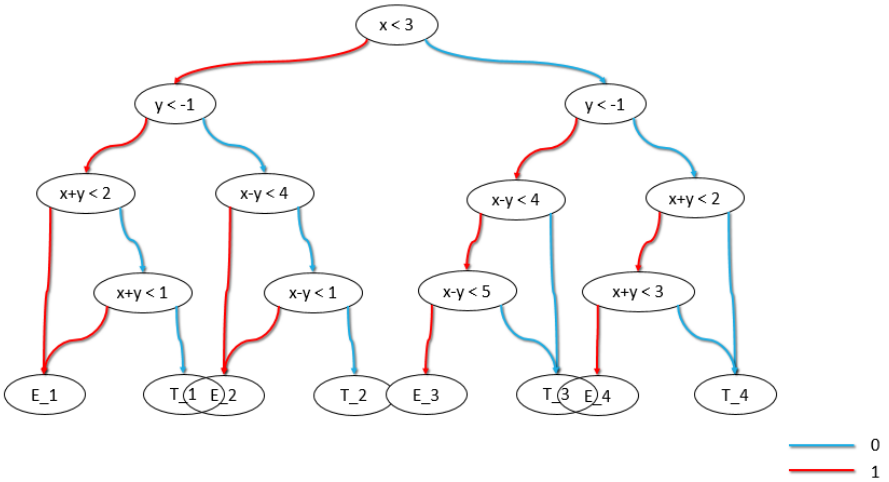

FIGURE 3.12: Représentation générale des 4 cas

Chapitre 4

Conclusion générale

Dans ce rapport, nous vous avons présenté un nouveau type de représentation abstraite qui s'inspire des diagrammes de décisions linéaires présenté en [CGS09]. En effet, les LDD et les LDD$^+$ se ressemblent de par leurs représentations sous forme de polyèdre, possèdent des règles de réduction identiques pour les cas ou les nœuds sont identiques, l'implication forte ou faible, etc. L'ordonnancement des LDD$^+$ est quand même différente des LDD du fait de ses variables variant jusqu'à l'infini mais nous n'avons pas abordé cette question afin d'y trouver une solution pour l'instant. Le travail effectué durant le stage consiste à inclure de nouvelles combinaisons dans l'arbre LDD si nécessaire afin de le réduire au mieux.

Bilan du stage

Effectuer un stage sur les LDD$^+$ était en quelque sorte une continuation des cours dispensés en Master 1 et Master 2 ESTel sur les BDDs et les langages synchrones. C'était d'ailleurs sur les BDDs que s'appuyait la construction des LDDs$^+$, il m'a fallu du temps pour apprivoiser la bibliothèque BDDs, savoir où regarder, etc. J'espère pouvoir apporter encore plus au monde de la recherche qui, je trouve, est un monde de passionné.

J'ai été très heureuse de faire ce stage à Inria. C'est un cadre de travail enrichissant et diversifié avec des chercheurs de spécialités et nationalités diverses. Ma petite contribution à ce gros projet qu'est NeuComp, à travers les diagrammes de décision linéaires, m'a aussi fait découvrir comment peuvent travailler autant de personnes sur un seul et même projet, surtout aussi vaste. Je me suis surpassée et vaincu mes peurs par rapport à la programmation grâce à ce premier contact avec le milieu de l'informatique professionnelle, et de la recherche. Tant au niveau professionnel que personnel j'y ai beaucoup appris.

Ce stage m'a permis aussi de faire le point sur mon orientation, et de choisir en toute connaissance de cause mon orientation vers l'Enseignement et la Recherche.

Annexe A

Démonstration $34! = 0$ sur 32 bits

$$
\begin{aligned}
2! &= 2 \\
3! &= 3 \bullet 2! = 3 \bullet 2 \\
4! &= 4 \bullet 3! = 2^2 \bullet (3 \bullet 2) \\
&= 3 \bullet 2^3 \\
5! &= 5 \bullet 4! = 5 \bullet (3 \bullet 2^3) \\
&= 5 \bullet 3 \bullet 2^3 \\
6! &= 6 \bullet 5! = 3 \bullet 2 \bullet (5 \bullet 3 \bullet 2^3) \\
&= 5 \bullet 3^2 \bullet 2^4 \\
7! &= 7 \bullet 6! = 7 \bullet (5 \bullet 3^2 \bullet 2^4) \\
&= 7 \bullet 5 \bullet 3^2 \bullet 2^4 \\
8! &= 8 \bullet 7! = 2^3 \bullet (7 \bullet 5 \bullet 3^2 \bullet 2^4) \\
&= 7 \bullet 5 \bullet 3^2 \bullet 2^7 \\
9! &= 9 \bullet 8! = 3^2 \bullet (7 \bullet 5 \bullet 3^2 \bullet 2^7) \\
&= 7 \bullet 5 \bullet 3^4 \bullet 2^7 \\
10! &= 10 \bullet 9! = 5 \bullet 2 \bullet (7 \bullet 5 \bullet 3^4 \bullet 2^7) \\
&= 7 \bullet 5^2 \bullet 3^4 \bullet 2^8 \\
11! &= 11 \bullet 10! = 11 \bullet (7 \bullet 5^2 \bullet 3^4 \bullet 2^8) \\
&= 11 \bullet 7 \bullet 5^2 \bullet 3^4 \bullet 2^8 \\
12! &= 12 \bullet 11! = 3 \bullet 2^2 \bullet (11 \bullet 7 \bullet 5^2 \bullet 3^4 \bullet 2^8) \\
&= 11 \bullet 7 \bullet 5^2 \bullet 3^5 \bullet 2^{10} \\
13! &= 13 \bullet 12! = 13 \bullet (11 \bullet 7 \bullet 5^2 \bullet 3^5 \bullet 2^{10}) \\
&= 13 \bullet 11 \bullet 7 \bullet 5^2 \bullet 3^5 \bullet 2^{10} \\
14! &= 14 \bullet 13! = 7 \bullet 2 \bullet (13 \bullet 11 \bullet 7 \bullet 5^2 \bullet 3^5 \bullet 2^{10}) \\
&= 13 \bullet 11 \bullet 7^2 \bullet 5^2 \bullet 3^5 \bullet 2^{11} \\
15! &= 15 \bullet 14! = 5 \bullet 3 \bullet (13 \bullet 11 \bullet 7^2 \bullet 5^2 \bullet 3^5 \bullet 2^{11}) \\
&= 13 \bullet 11 \bullet 7^2 \bullet 5^3 \bullet 3^6 \bullet 2^{11} \\
16! &= 16 \bullet 15! = 2^4 \bullet (13 \bullet 11 \bullet 7^2 \bullet 5^3 \bullet 3^6 \bullet 2^{11}) \\
&= 13 \bullet 11 \bullet 7^2 \bullet 5^3 \bullet 3^6 \bullet 2^{15} \\
17! &= 17 \bullet 16! = 17 \bullet (13 \bullet 11 \bullet 7^2 \bullet 5^3 \bullet 3^6 \bullet 2^{15}) \\
&= 17 \bullet 13 \bullet 11 \bullet 7^2 \bullet 5^3 \bullet 3^6 \bullet 2^{15}
\end{aligned}
$$

$$
\begin{aligned}
18! = \quad 18 \bullet 17! &= 3^2 \bullet 2 \bullet (17 \bullet 13 \bullet 11 \bullet 7^2 \bullet 5^3 \bullet 3^6 \bullet 2^{15}) \\
&= 17 \bullet 13 \bullet 11 \bullet 7^2 \bullet 5^3 \bullet 3^8 \bullet 2^{16} \\
19! = \quad 19 \bullet 18! &= 19 \bullet (17 \bullet 13 \bullet 11 \bullet 7^2 \bullet 5^3 \bullet 3^8 \bullet 2^{16}) \\
&= 19 \bullet 17 \bullet 13 \bullet 11 \bullet 7^2 \bullet 5^3 \bullet 3^8 \bullet 2^{16} \\
20! = \quad 20 \bullet 19! &= 5 \bullet 2^2 \bullet (19 \bullet 17 \bullet 13 \bullet 11 \bullet 7^2 \bullet 5^3 \bullet 3^8 \bullet 2^{16}) \\
&= 19 \bullet 17 \bullet 13 \bullet 11 \bullet 7^2 \bullet 5^4 \bullet 3^8 \bullet 2^{18} \\
21! = \quad 21 \bullet 20! &= 7 \bullet 3 \bullet (19 \bullet 17 \bullet 13 \bullet 11 \bullet 7^2 \bullet 5^4 \bullet 3^8 \bullet 2^{18}) \\
&= 19 \bullet 17 \bullet 13 \bullet 11 \bullet 7^3 \bullet 5^4 \bullet 3^9 \bullet 2^{18} \\
22! = \quad 22 \bullet 21! &= 11 \bullet 2 \bullet (19 \bullet 17 \bullet 13 \bullet 11 \bullet 7^3 \bullet 5^4 \bullet 3^9 \bullet 2^{18}) \\
&= 19 \bullet 17 \bullet 13 \bullet 11^2 \bullet 7^3 \bullet 5^4 \bullet 3^9 \bullet 2^{19} \\
23! = \quad 23 \bullet 22! &= 23 \bullet (19 \bullet 17 \bullet 13 \bullet 11^2 \bullet 7^3 \bullet 5^4 \bullet 3^9 \bullet 2^{19}) \\
&= 23 \bullet 19 \bullet 17 \bullet 13 \bullet 11^2 \bullet 7^3 \bullet 5^4 \bullet 3^9 \bullet 2^{19} \\
24! = \quad 24 \bullet 23! &= 3 \bullet 2^3 \bullet (23 \bullet 19 \bullet 17 \bullet 13 \bullet 11^2 \bullet 7^3 \bullet 5^4 \bullet 3^9 \bullet 2^{19}) \\
&= 23 \bullet 19 \bullet 17 \bullet 13 \bullet 11^2 \bullet 7^3 \bullet 5^4 \bullet 3^{10} \bullet 2^{22} \\
25! = \quad 25 \bullet 24! &= 5^2 \bullet (23 \bullet 19 \bullet 17 \bullet 13 \bullet 11^2 \bullet 7^3 \bullet 5^4 \bullet 3^{10} \bullet 2^{22}) \\
&= 23 \bullet 19 \bullet 17 \bullet 13 \bullet 11^2 \bullet 7^3 \bullet 5^6 \bullet 3^{10} \bullet 2^{22} \\
26! = \quad 26 \bullet 25! &= 13 \bullet 2 \bullet (23 \bullet 19 \bullet 17 \bullet 13 \bullet 11^2 \bullet 7^3 \bullet 5^6 \bullet 3^{10} \bullet 2^{22}) \\
&= 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^3 \bullet 5^6 \bullet 3^{10} \bullet 2^{23} \\
27! = \quad 27 \bullet 26! &= 3^3 \bullet (23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^3 \bullet 5^6 \bullet 3^{10} \bullet 2^{23}) \\
&= 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^3 \bullet 5^6 \bullet 3^{13} \bullet 2^{23} \\
28! = \quad 28 \bullet 27! &= 7 \bullet 2^2 \bullet (23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^3 \bullet 5^6 \bullet 3^{13} \bullet 2^{23}) \\
&= 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^4 \bullet 5^6 \bullet 3^{13} \bullet 2^{25} \\
29! = \quad 29 \bullet 28! &= 29 \bullet (23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^4 \bullet 5^6 \bullet 3^{13} \bullet 2^{25}) \\
&= 29 \bullet 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^4 \bullet 5^6 \bullet 3^{13} \bullet 2^{25} \\
30! = \quad 30 \bullet 29! &= 5 \bullet 3 \bullet 2 \bullet (29 \bullet 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^4 \bullet 5^6 \bullet 3^{13} \bullet 2^{25}) \\
&= 29 \bullet 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^4 \bullet 5^7 \bullet 3^{14} \bullet 2^{26} \\
31! = \quad 31 \bullet 30! &= 31 \bullet (29 \bullet 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^4 \bullet 5^7 \bullet 3^{14} \bullet 2^{26}) \\
&= 31 \bullet 29 \bullet 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^4 \bullet 5^7 \bullet 3^{14} \bullet 2^{26} \\
32! = \quad 32 \bullet 31! &= 2^5 \bullet (31 \bullet 29 \bullet 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^4 \bullet 5^7 \bullet 3^{14} \bullet 2^{26}) \\
&= 31 \bullet 29 \bullet 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^4 \bullet 5^7 \bullet 3^{14} \bullet 2^{31} \\
33! = \quad 33 \bullet 32! &= 11 \bullet 3 \bullet (31 \bullet 29 \bullet 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^2 \bullet 7^4 \bullet 5^7 \bullet 3^{14} \bullet 2^{31}) \\
&= 31 \bullet 29 \bullet 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^3 \bullet 7^4 \bullet 5^7 \bullet 3^{15} \bullet 2^{31} \\
34! = \quad 34 \bullet 33! &= 17 \bullet 2 \bullet (31 \bullet 29 \bullet 23 \bullet 19 \bullet 17 \bullet 13^2 \bullet 11^3 \bullet 7^4 \bullet 5^7 \bullet 3^{15} \bullet 2^{31}) \\
&= 31 \bullet 29 \bullet 23 \bullet 19 \bullet 17^2 \bullet 13^2 \bullet 11^3 \bullet 7^4 \bullet 5^7 \bullet 3^{15} \bullet 2^{32} \\
&= N \bullet 2^{32} \\
&= N << 32
\end{aligned}
$$

avec $N = 31 \bullet 29 \bullet 23 \bullet 19 \bullet 17^2 \bullet 13^2 \bullet 11^3 \bullet 7^4 \bullet 5^7 \bullet 3^{15}$

or 34! est codé sur 32 bits. Le décalage à gauche de N va insérer 32 zeros qui vont finalement remplir complètement la zone mémoire de 32 bits allouée à ce nombre. Donc $34! = 0$ sur 32 bits. CQFD

Bibliographie

[Abd14] Mariem Abdelmoula. *Génération automatique de jeux de tests avec analyse symbolique des données pour les systèmes embarqués. (Automatic generation of tests with data symbolic analysis for the embedded systems)*. PhD thesis, University of Nice Sophia Antipolis, France, 2014.

[Arn91] Douglas N. Arnold. The patriot missile failure, 1991. http ://www-users.math.umn.edu/ arnold/disasters/patriot.html.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. pages 495–496, September August 1986.

[CGS09] S. Chaki, A. Gurfinkel, and O. Strichman. Decision diagrams for linear arithmetic. pages 53–60, Nov 2009.

[Cou78] Patrick Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes*. Habilitation à diriger des recherches, Institut National Polytechnique de Grenoble - INPG ; Université Joseph-Fourier - Grenoble I, March 1978. Universités : Université scientifique et médicale de Grenoble et Institut national polytechnique de Grenoble.

[DMMG+16] Elisabetta De Maria, Alexandre Muzy, Daniel Gaffé, Annie Ressouche, and Franck Grammont. Verification of temporal properties of neuronal archetypes modeled as synchronous reactive systems. LNBI 9957 :1–16, October 2016.

[GR13] Daniel Gaffé and Annie Ressouche. Algebraic framework for synchronous language semantics. In *Theoretical Aspects of Software Engineering (TASE), 2013 International Symposium on*, pages 51–58, Birmingham, UK, July 1-3 2013. IEEE.

[INR] INRIA. Institut nationale de recherche en informatique et mathématique. https ://www.inria.fr.

[Kri63] Saul A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16 :83–94, 1963.

[La96] J. L. Lions and al. Ariane 5, flight 501 failure, report by the inquiry board, 1996. http ://sunnyday.mit.edu/accidents/Ariane5accidentreport.html.

[Min04] Antoine Miné. *Weakly Relational Numerical Abstract Domains*. Theses, Ecole Polytechnique X, December 2004.

[MLAH99] Jesper Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. *Difference Decision Diagrams*, pages 111–125. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[PR76] Cousot P. and Cousot R. Static determination of dynamic properties of programs. Dunod, Paris, 1976.

[RGR08] Annie Ressouche, Daniel Gaffé, and Valérie Roy. Modular compilation of a synchronous language. *CoRR*, abs/0801.3715, 2008.